4. Dawn

Program Name: Dawn.java Test Input File: dawn.dat

Dawn has seen news stories about people winning big lottery drawings with the option to receive either a single lump sum payment or a series of payments over several years from an annuity. She wondered how such an annuity would work if she were to receive a large inheritance from a long-lost relative, "Uh sure," she thought to herself. But, she decided this would be an opportunity to use her programming skills.

During her research, she discovered several options for annuity payments and settled on the following formula, calculating a fixed monthly payment for an annuity with a fixed annual percentage rate (APR) and no further deposits toward the initial investment.

Formula	Legend

$\boldsymbol{p = \frac{ar}{1-(1+r)^{-n}}}$	p = amount of payment
	a = amount invested
	r = monthly interest rate
	n = number of payments

The investment amount is standard currency with dollars and cents and the amount could reach $250,000.00. The monthly interest rate is one-twelfth the APR as a decimal value with the APR provided as a percentage like 5.995%. The length of the annuity is the number of years, a whole number, for which Dawn will receive monthly payments.

Dawn is aware of floating-point issues in programming and recalls her computer science teacher telling them when working with money, calculated amounts must be rounded to the nearest $0.01 before using them in subsequent computations; otherwise, partial pennies eventually turn into whole pennies causing later computations to produce incorrect values. In order to test her handling of money values, Dawn decides she will produce some totals to check her rounding.

Input: The first line of the data file contains a count of the number of data sets, with a max of 20 to prevent totals from exceeding the field widths shown below. Each set of data will contain a line with three floating-point values: amount invested, annual percentage rate (APR), and number of years. Whitespace separates all values.

Output: A table containing the following items, formatted exactly as shown, **with alignment dots** in the second line of column headers and vertical lines "|" between columns. The last line contains totals for those columns. Input values and calculated results will not exceed the field widths shown below.

```
     Amount                             Monthly         Total of
....Invested........APR......Years......Payment.........Payments............Profit..
$  130,360.20  |   6.825%  |   36  |  $    811.44  |  $  350,542.08  |  $  220,181.88
$  131,548.95  |  10.020%  |   25  |  $ 1,197.24  |  $  359,172.00  |  $  227,623.05
$  154,472.61  |   3.657%  |   12  |  $ 1,326.87  |  $  191,069.28  |  $   36,596.67
$  116,466.55  |   3.478%  |   39  |  $    454.98  |  $  212,930.64  |  $   96,464.09
$  532,848.31  |           |       |  $ 3,790.53  |  $1,113,714.00  |  $  580,865.69
```

Sample input:

```
4
130360.20    6.825      36
131548.95   10.020      25
154472.61    3.657      12
116466.55    3.478      39
```

Sample output: The exact table with precise formatting as shown above but without the outside border lines.